UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/767,480	01/28/2004	Norman Rubin	00100.03.0040	5073

29153 7590 12/30/2008

ADVANCED MICRO DEVICES, INC.
C/O VEDDER PRICE P.C.
222 N.LASALLE STREET
CHICAGO, IL 60601

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
12/30/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/767,480	RUBIN ET AL.	
	Examiner	Art Unit	
	BEN C. WANG	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *01 October 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-9, 11-16, and 18-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-5,8,9,11,14-16,19 and 21-24* is/are rejected.

7)☒ Claim(s) *6,7,12,13,18 and 20* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *9/19/2008*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated October 1, 2008, responding to the Office action mailed June 10, 2008 provided in the rejection of claims 1-9, 11-16 and 18-24, wherein claims 1, 8, 14 and 19 are amended.

Claims 1-9, 11-16 and 18-24 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are moot in view of the new grounds of rejection – see *Nair* et al. - art made of record, as applied hereto.

2. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Information Disclosure Statement

3. The information disclosure statement (IDS) submitted on 19 September 2008 was filed after the mailing date of the Office action on 10 June 2008. The submission is in compliance with the provisions of 37 CFR 1.97. Accordingly, the information disclosure statement is being considered by the examiner.

Claim Rejections – 35 USC § 102(e)

The following is quotation of 35 U.S.C. 102(e) which form the basis for all obviousness rejections set forth in this office action:

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

4. Claims 1-5, 8-9, 11, 14-16, 19, and 21-24 are rejected under 35 U.S.C. 102(e) as being anticipated by Nair et al. (Pat. No. US 7,353,503 B2) (hereinafter 'Nair' - art made of record)

5. **As to claim 1** (Currently Amended), Nair discloses A method for static single assignment form dead code elimination (e.g., Abstract – a method for eliminating dead code from a computer program using an operands graph generated from a flow graph of a computer program; Col. 5, Lines 55-62 - ... any Static Single Assignment (SSA) algorithm may be used to generate operands graph; Col. 10, Lines 38-57), the method comprising:

- examining a first instruction of a machine code off of a worklist in memory wherein the first instruction includes a previous link and a write mask (e.g., Col. 2, Lines 42-63 - ... efficient dead code elimination ... uses an operands as a mechanism by which to identify unused code (i.e., dead instructions). An operands graph for a computer program is *a linked list* (i.e., a worklist) of all the operands defined and used within the computer program. The operand graph includes information ... a pointer to the instruction that defined the operand, and a count of the number of times the operand is used within a computer program (an operand reference count) ... an operand refers to a parameter (often represented by a register) of an assembly language instruction ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution; Fig. 4a, element 402(e) – *ptr* (i.e., previous link); Col. 5, Lines 63-67 – Fig. 4a illustrate an exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including an operand type 402(a), and operand reference count 402(b) and a pointer to the instruction which defined the operands 402(c); Col. 3, Lines 9-19 – To create the operands graph, a unique memory location (e.g., a virtual register) is created for each operand that is defined in a program (i.e., in the flow graph). This virtual register is used to store the operand type, the operand reference count and the defining

instruction pointer (i.e., a write mask) ...; Col. 7, Lines 32-37 – In another embodiment of the present invention, an operand is marked dead by tagging the virtual register associated with the operand with a specific value (e.g., 'true') (i.e., a write mask) ...);

- examining at least one second instruction, wherein the at least one second instruction is a source of the first instruction and wherein each of the at least one second instruction includes a previous link and a write mask (e.g., Fig. 4b; Col. 6, Lines 1-19 – Fig. 4b illustrates how operands graph 400 represents operand information fro operands in routine R1 of program 302 ... The pointer to the defining instruction for operand type B1 is '*ptr*' to reflect that r1 is defined outside the scope of routine R1. It will be recognized that the present invention can be used to generate an operands graph for any scope of program 302);
- determining, using said previous link of said at least one second instruction, if any components within a particular field of the at least one second instruction are required, wherein said previous link of said second instruction links said second instruction with a prior instruction that writes at least one component of said components (e.g., Col. 2, Lines 42-63 - ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... Further ... an unused (i.e., dead) operand is an operand that has been defined, but is not used, either within the computer program as a whole, or a smaller scope of the computer program ... Identifying unused instructions ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the

instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution);

- when no components of the at least one second instruction are required, deleting the at least one second instruction from the machine code (e.g., Fig. 5; Col. 6, Lines 57-63 – The clean-up involves removing each of the dead instructions in the routine (step 514), adjusting the linked list in the basic block to compensate for the removed instructions (step 516), removing any basic blocks for which all of the instructions have been removed (step 518), and adjusting the operands graph to compensate for any basic blocks removed (520)); and
- when any component of the at least one second instruction is required, adding the at least one second instruction to the worklist in the memory (e.g., Fig. 3c; Fig. 4a; ; Col. 5, Lines 37-67 – With respect to FIG. 3c. a link list representation of flow graph 300 is presented … includes a linked list of routines, a linked list of basic blocks, a linked list of edges, and a linked list of operands … Each operand within linked list of operands represents an operand within program 302 … Fig. 4a illustrates an exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including … a pointer to the instruction which defined the operand 402(c))

6. **As to claim 2** (original) (incorporating the rejection in claim 1), Nair discloses the method further comprising: generating the worklist by:

- for each of a plurality of instructions, determining if the instruction is a critical instruction; and

- if the instruction is a critical instruction, writing the instruction to the worklist (e.g., (e.g., Fig. 4b; Col. 6, Lines 1-19 – Fig. 4b illustrates how operands graph 400 represents operand information fro operands in routine R1 of program 302 … The pointer to the defining instruction for operand type B1 is '*ptr*' to reflect that r1 is defined outside the scope of routine R1. It will be recognized that the present invention can be used to generate an operands graph for any scope of program 302)

7. **As to claim 3** (original) (incorporating the rejection in claim 2), Nair discloses the method further comprising: setting a live bit for each component of the plurality of instructions (e.g., Col. 3, Lines 9-19 – To create the operands graph, a unique memory location (e.g., a virtual register) is created for each operand that is defined in a program (i.e., in the flow graph). This virtual register is used to store the operand type, the operand reference count and the defining instruction pointer (i.e., a write mask) ...; Col. 7, Lines 32-37 – In another embodiment of the present invention, an operand is marked dead by tagging the virtual register associated with the operand with a specific value (e.g., 'true') (i.e., a write mask) …)

8. **As to claim 4** (Previously Presented) (incorporating the rejection in claim 2), Nair discloses the method wherein each critical instruction is an instruction that generates an export value (e.g., Col. 3, Lines 9-19 – To create the operands graph, a unique memory location (e.g., a virtual register) is created for each operand that is defined in a program

(i.e., in the flow graph). This virtual register is used to store the operand type, the operand reference count and the defining instruction pointer (i.e., a write mask) ...; Col. 7, Lines 32-37 – In another embodiment of the present invention, an operand is marked dead by tagging the virtual register associated with the operand with a specific value (e.g., 'true') (i.e., a write mask) ...)

9. **As to claim 5** (Previously Presented) (incorporating the rejection in claim 2), Nair discloses the method further comprising: prior to generating the worklist: receiving a plurality of instructions; adding to each instruction the previous link (e.g.,); and adding to each instruction the write mask (e.g., Col. 2, Lines 42-63 - ... efficient dead code elimination ... uses an operands as a mechanism by which to identify unused code (i.e., dead instructions). An operands graph for a computer program is *a linked list* (i.e., a worklist) of all the operands defined and used within the computer program. The operand graph includes information ... a pointer to the instruction that defined the operand, and a count of the number of times the operand is used within a computer program (an operand reference count) ... an operand refers to a parameter (often represented by a register of an assembly language instruction ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution; Fig. 4a, element 402(e) – *ptr* (i.e., previous link); Col. 5, Lines 63-67 – Fig. 4a illustrate an exemplary operands graph 400.

Operands graph includes a linked list of operands 402, with each operand 402 including an operand type 402(a), and operand reference count 402(b) and a pointer to the instruction which defined the operands 402(c); Col. 3, Lines 9-19 – To create the operands graph, a unique memory location (e.g., a virtual register) is created for each operand that is defined in a program (i.e., in the flow graph). This virtual register is used to store the operand type, the operand reference count and the defining instruction pointer (i.e., a write mask) ...; Col. 7, Lines 32-37 – In another embodiment of the present invention, an operand is marked dead by tagging the virtual register associated with the operand with a specific value (e.g., 'true') (i.e., a write mask) ...)

10. **As to claim 8** (Currently Amended), Nair discloses a method for static single assignment form dead code elimination (e.g., Abstract – a method for eliminating dead code from a computer program using an operands graph generated from a flow graph of a computer program; Col. 5, Lines 55-62 - ... any Static Single Assignment (SSA) algorithm may be used to generate operands graph; Col. 10, Lines 38-57) comprising:

- receiving a plurality of instructions;;
- adding to each instruction a previous link, wherein said previous link links said each instruction with a prior instruction that writes at least one component;
- adding to each instruction a write mask (e.g., Col. 2, Lines 42-63 - ... efficient dead code elimination ... uses an operands as a mechanism by which to identify unused code (i.e., dead instructions). An operands graph for a computer program is *a linked list* (i.e., a worklist) of all the operands defined and used within the

computer program. The operand graph includes information … a pointer to the instruction that defined the operand, and a count of the number of times the operand is used within a computer program (an operand reference count) … an operand refers to a parameter (often represented by a register (i.e., a write mask)) of an assembly language instruction … an operand is defined when the operand is the target (e.g., stores the result) of such an instruction … involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution; Fig. 4a, element 402(e) – ptr (i.e., previous link); Col. 5, Lines 63-67 – Fig. 4a illustrate an exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including an operand type 402(a), and operand reference count 402(b) and a pointer to the instruction which defined the operands 402(c)); and

generating a worklist in memory by:

- for each of the plurality of instructions, determining if the instruction is a critical instruction (e.g., Fig. 5; Col. 6, Lines 57-63 – The clean-up involves removing each of the dead instructions in the routine (step 514), adjusting the linked list in the basic block to compensate for the removed instructions (step 516), removing any basic blocks for which all of the instructions have been removed (step 518), and adjusting the operands graph to compensate for any basic blocks removed (520)); and

- if the instruction is a critical instruction, writing the instructions to the worklist in the memory (e.g., Fig. 3c; Fig. 4a; ; Col. 5, Lines 37-67 – With respect to FIG. 3c. a link list representation of flow graph 300 is presented … includes a linked list of routines, a linked list of basic blocks, a linked list of edges, and a linked list of operands … Each operand within linked list of operands represents an operand within program 302 … Fig. 4a illustrates an exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including … a pointer to the instruction which defined the operand 402(c))

11. **As to claim 9** (Previously Presented) (incorporating the rejection in claim 8), Nair discloses the method of claim 8 further comprising: setting a live bit for each component of the plurality of instructions:

- examining a first instruction off of the worklist (e.g., Abstract – a method for eliminating dead code from a computer program using an operands graph generated from a flow graph of a computer program);
- examining at least one second instruction in the machine code, wherein the at least one instruction is a source of the first instruction (e.g., Fig. 3c; Fig. 4a; ; Col. 5, Lines 37-67 – With respect to FIG. 3c. a link list representation of flow graph 300 is presented … includes a linked list of routines, a linked list of basic blocks, a linked list of edges, and a linked list of operands … Each operand within linked list of operands represents an operand within program 302 … Fig. 4a illustrates

an exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including ... a pointer to the instruction which defined the operand 402(c));

- determining if any component within a particular field of the at least one second instruction is live (e.g., Col. 2, Lines 42-63 - ... ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution); and
- when no components of the at least one second instruction are live, deleting the second instruction from the worklist (e.g., Fig. 5; Col. 6, Lines 57-63 – The clean-up involves removing each of the dead instructions in the routine (step 514), adjusting the linked list in the basic block to compensate for the removed instructions (step 516), removing any basic blocks for which all of the instructions have been removed (step 518), and adjusting the operands graph to compensate for any basic blocks removed (520))

12. **As to claim 11** (Previously Presented) (incorporating the rejection in claim 8), please refer to claim **4** above, accordingly.

13. **As to claim 14** (Currently Amended), Nair discloses an apparatus for static single assignment form dead code elimination (e.g., Abstract – a method for eliminating

dead code from a computer program using an operands graph generated from a flow graph of a computer program; Col. 5, Lines 55-62 - ... any Static Single Assignment (SSA) algorithm may be used to generate operands graph; Col. 10, Lines 38-57) comprising:

at least one memory device storing a plurality of executable instructions of a machine code; and

at least one processor operably coupled to the at least one memory device, operative to receive the plurality of executable instructions such that the at least one processor, in response to plurality of executable instructions is further operative to:

- examine a first instruction off of a worklist, wherein the first instruction includes previous link and a write mask (e.g., Col. 2, Lines 42-63 - ... efficient dead code elimination ... uses an operands as a mechanism by which to identify unused code (i.e., dead instructions). An operands graph for a computer program is *a linked list* (i.e., a worklist) of all the operands defined and used within the computer program. The operand graph includes information ... a pointer to the instruction that defined the operand, and a count of the number of times the operand is used within a computer program (an operand reference count) ... an operand refers to a parameter (often represented by a register (i.e., a write mask)) of an assembly language instruction ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction

that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution; Fig. 4a, element 402(e) – ptr (i.e., previous link); Col. 5, Lines 63-67 – Fig. 4a illustrate an exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including an operand type 402(a), and operand reference count 402(b) and a pointer to the instruction which defined the operands 402(c));

- examine at least one second instruction of the machine code, wherein the at least one second instruction is a source of the first instruction and each of the at least one second instruction includes a previous link and a write mask (e.g., Fig. 4b; Col. 6, Lines 1-19 – Fig. 4b illustrates how operands graph 400 represents operand information fro operands in routine R1 of program 302 … The pointer to the defining instruction for operand type B1 is '*ptr*' to reflect that r1 is defined outside the scope of routine R1. It will be recognized that the present invention can be used to generate an operands graph for any scope of program 302);
- determine, using said previous link of said at least one second instruction, if any component within a particular field of the at least one second instruction is live, wherein said previous link of said second instruction links said second instruction with a prior instruction that writes at least one component of said components (e.g., Col. 2, Lines 42-63 - … … an operand is defined when the operand is the target (e.g., stores the result) of such an instruction … involves

a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution); and

- when no components are live, delete the second instruction from the machine code (e.g., Fig. 5; Col. 6, Lines 57-63 – The clean-up involves removing each of the dead instructions in the routine (step 514), adjusting the linked list in the basic block to compensate for the removed instructions (step 516), removing any basic blocks for which all of the instructions have been removed (step 518), and adjusting the operands graph to compensate for any basic blocks removed (520))

14. **As to claim 15** (Previously Presented) (incorporating the rejection in claim 14), please refer to claim **2** above, accordingly.

15. **As to claim 16** (Previously Presented) (incorporating the rejection in claim 15), please refer to claim **4** above, accordingly.

16. **As to claim 19** (Currently Amended), Nair discloses an apparatus for static single assignment form dead code eliminations (e.g., Abstract – a method for eliminating dead code from a computer program using an operands graph generated from a flow graph of a computer program; Col. 5, Lines 55-62 - … any Static Single Assignment (SSA) algorithm may be used to generate operands graph; Col. 10, Lines

38-57) comprising: at least one memory device storing a plurality of executable instructions of a machine code; and at least one processor operably coupled to the at least one memory device, operative to receive the plurality of executable instructions such that the at least one processor, in response to the executable instructions is further operative to:

- receive a plurality of instructions;
- add to each instruction a previous link, wherein said previous link links said each instruction with a prior instruction that writes at least one component;
- add to each instruction a write mask (e.g., Col. 2, Lines 42-63 - ... efficient dead code elimination ... uses an operands as a mechanism by which to identify unused code (i.e., dead instructions). An operands graph for a computer program is *a linked list* (i.e., a worklist) of all the operands defined and used within the computer program. The operand graph includes information ... a pointer to the instruction that defined the operand, and a count of the number of times the operand is used within a computer program (an operand reference count) ... an operand refers to a parameter (often represented by a register (i.e., a write mask)) of an assembly language instruction ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution; Fig. 4a, element 402(e) – ptr (i.e., previous link); Col. 5, Lines 63-67 – Fig. 4a illustrate an

exemplary operands graph 400. Operands graph includes a linked list of operands 402, with each operand 402 including an operand type 402(a), and operand reference count 402(b) and a pointer to the instruction which defined the operands 402(c)); and

generate a worklist by:

- for each of the plurality of instructions; determining if the instruction is a critical instruction; and
- if the instruction is a critical instruction, writing the instructions to the worklist; examine a first instruction off of the worklist (e.g., Fig. 4b; Fig. 4b illustrates how operands graph 400 represents operand information for operands in routine R1 of program 302 ... The pointer to the defining instruction for operand type B1 is 'ptr' to reflect that r11 is defined outside the scope of routine R1. It will be recognized that the present invention can be used to generate an operands graph for any scope of program 302 ...);
- examine at least one second instruction from the machine code, wherein the at least one second instruction is a source of the first instruction (e.g., Fig. 4b; Col. 6, Lines 1-19 – Fig. 4b illustrates how operands graph 400 represents operand information fro operands in routine R1 of program 302 ... The pointer to the defining instruction for operand type B1 is '*ptr*' to reflect that r1 is defined outside the scope of routine R1. It will be recognized that the present invention can be used to generate an operands graph for any scope of program 302);

- determine, using a previous link of said at least one second instruction, if any component within a particular field of the at least one second instruction is live (e.g., Fig. 5; Col. 6, Lines 27-63 – Fig. 5 illustrates a flow chart 500 of a process performed by dead code elimination module 204 to eliminate dead code … The process continues recursively until each operand within the operands graph is evaluated ...) ; and
- when no component is live, delete the second instruction from the machine code (e.g., Fig. 5; Col. 6, Lines 57-63 – The clean-up involves removing each of the dead instructions in the routine (step 514), adjusting the linked list in the basic block to compensate for the removed instructions (step 516), removing any basic blocks for which all of the instructions have been removed (step 518), and adjusting the operands graph to compensate for any basic blocks removed (520))

17. **As to claim 21** (Previously Presented) (incorporating the rejection in claim 15), Nair discloses the apparatus wherein the at least one processor in response to the plurality of executable instructions is further operative to set a live bit for each component of the plurality of instructions (e.g., Col. 2, Lines 42-63 - … … an operand is defined when the operand is the target (e.g., stores the result) of such an instruction … involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution)

18. **As to claim 22** (Previously Presented) (incorporating the rejection in claim 9), Nair discloses the method further comprising: when any component of the at least one second instruction is live , adding the at least one second instruction to the worklist (e.g., Col. 2, Lines 42-63 - ... ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution)

19. **As to claim 23** (Previously Presented) (incorporating the rejection in claim 14), Nair discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: when any component of the at least one second instruction is live, add the at least one second instruction to the worklist (e.g., Col. 2, Lines 42-63 - ... ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution)

20. **As to claim 24** (Previously Presented) (incorporating the rejection in claim 19), Nair discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: when any

component of the at least one second instruction is live, add the at least one second instruction to the worklist (e.g., Col. 2, Lines 42-63 - ... ... an operand is defined when the operand is the target (e.g., stores the result) of such an instruction ... involves a process of searching the operands graph for unused operands, obtaining a pointer to the instruction that defined the unused operand, marking the defining instruction as dead, and removing the instruction from execution)

Allowable Subject Matter

21. Claims 6-7, 12-13, 18, and 20 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten to overcome all the limitations of the base claim and any intervening claims.

The following is an examiner's statement of reasons for allowance:

22. **Regarding claims** 6-7, 12-13, 18, and 20, prior art of record fails to reasonably show or suggest "the write mask is a multi-bit field representing a number of components in a superword register; each of the plurality of instructions are written to the worklist a predetermined number of times, wherein the predetermined number of times is based on the number of components in the superword register"

Conclusion

23. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Ben C. Wang
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192